Examining ASP.NET's Membership, Roles, and Profile - Part 3

By Scott Mitchell

Introduction

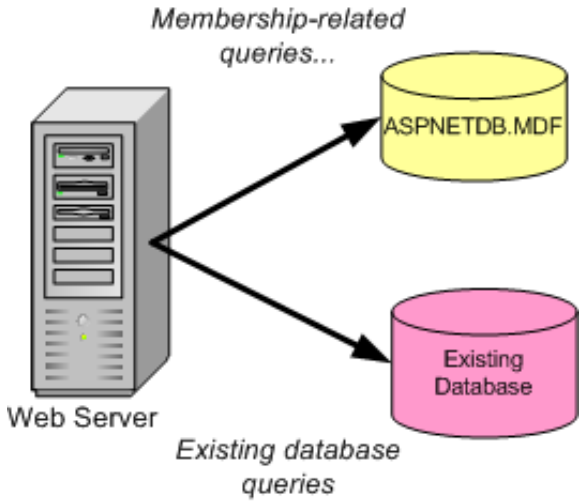
The membership and roles providers used by ASP.NET by default are the `SqlMembershipProvider` and `SqlRoleProvider`, respectively, which serialize membership and roles information to a SQL Server database. Specifically, this information is stored in a variety of pre-defined tables and accessed through a number of pre-defined stored procedures. In order to use membership or roles with your application using the default providers you need to apply this pre-defined schema to your application's database.

If you don't already have a database, ASP.NET will happily create a SQL Server 2005 Express Edition database in your application's `App_Data` folder that contains the necessary schema. While this is certainly handy, often developers are faced with a situation where they want to add ASP.NET's membership and roles features to an existing data model. For example, you may have an application that hasn't yet needed membership or roles services, but recent feature requests may require that certain areas of the site are available only to particular users, or certain functionality is limited to those who belong to a specific role. In any case, adding membership and roles support to an existing database can be accomplished through the ASP.NET SQL Server Registration Tool (`aspnet_regsql.exe`).

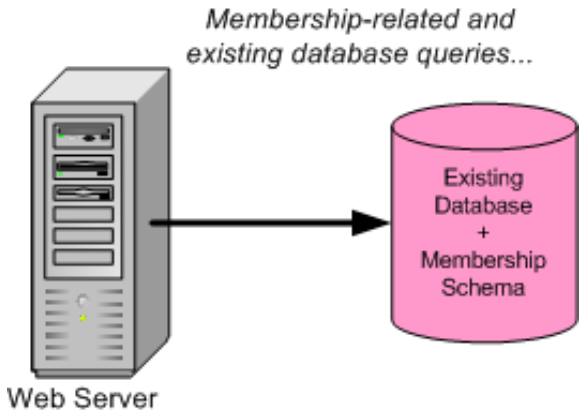
In this article we'll examine how to use this tool to apply the membership and roles services for the `SqlMembershipProvider` and `SqlRoleProvider` providers. This tool can be used both from the command-line and through the graphical wizard. Furthermore, we'll pay special attention on how to use this tool with an existing SQL Server 2005 Express Edition database in the `App_Data` folder, which can be a particularly challenging task. Read on to learn more!

Do You Need to Consolidate the Membership/Roles Schemas With Your Database?

As discussed in Part 1 of this article series, the ASP.NET Website Administration Tool (AWAT) will happily create a SQL Server 2005 Express Edition database named `ASPNETDB.MDF` for you in your application's `App_Data` folder that has the tables, views, and stored procedures needed to implement all of the membership-related services. And, without any changes, the default membership provider will use this database. In such a setting there are two databases in use here - `ASPNETDB.MDF` and your application's existsing database. Those queries related to membership and roles, such as authenticating a user's credentials, creating a new user account, and so on, will utilize the `ASPNETDB.MDF` database while your application's existing data-driven web pages will continue to use the existing database. The following diagram illustrates this workflow.



The idea is to move the membership-related schema into your application's existing database so that *all* database traffic is using the existing database. What we want instead is the following workflow:



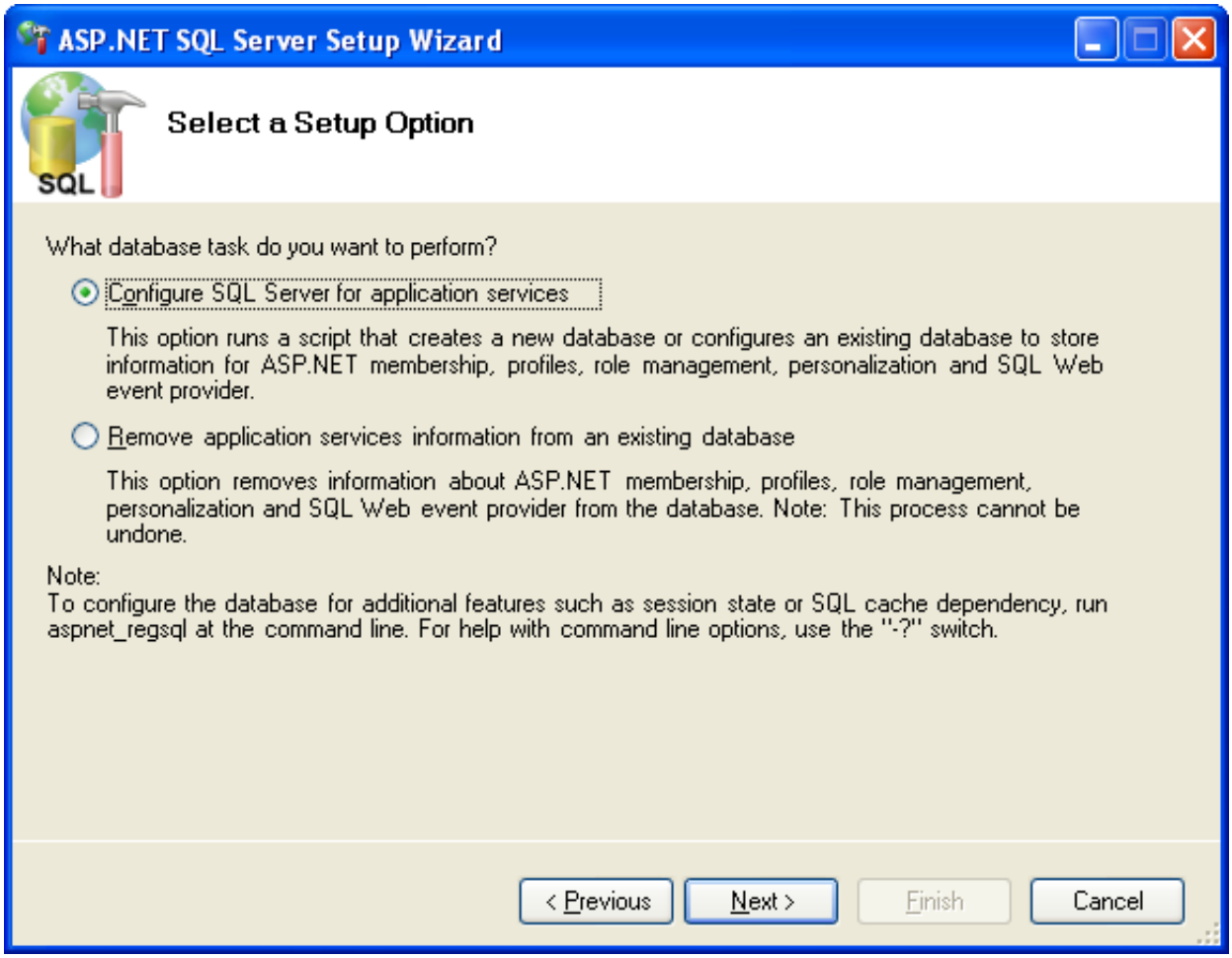
But is this consolidation needed? Why can't we have two different databases, one for membership services and one for the existing application's workload? If your application's existing tables or new tables will now or eventually need to reference membership information then the schemas should be merged to allow for foreign key constraints between your tables and the `aspnet_Users` or `aspnet_Roles` tables. If, however, you are using membership strictly for website-level access control and user information is not recorded in your application's data model in any manner, then it would be satisfactory to keep the schemas separate. However, I'd still encourage a melding of the schemas since requirements may change in the future, requiring information to be captured that relates to the logged on user.

For example, imagine that you have a photo album application site that currently allows only you to add photos by navigating to a "secret" URL. Rather than implementing security by obscurity, you decide you want to use ASP.NET's membership capabilities to only allow authenticated users to visit the URL to add images, with the idea being that there would be only one or two user accounts created for the system - you and your spouse, perhaps. In this case, having the membership schema external to the existing application database is probably satisfactory. However, if you wanted to add membership for your site to allow visitors to create accounts and leave comments and ratings for the pictures in your portfolio, then you'd want to include the membership schema in your application's database so that you can create a `Comments` database table that has a foreign key to the `aspnet_Users` table to indicate what registered user made the comment.

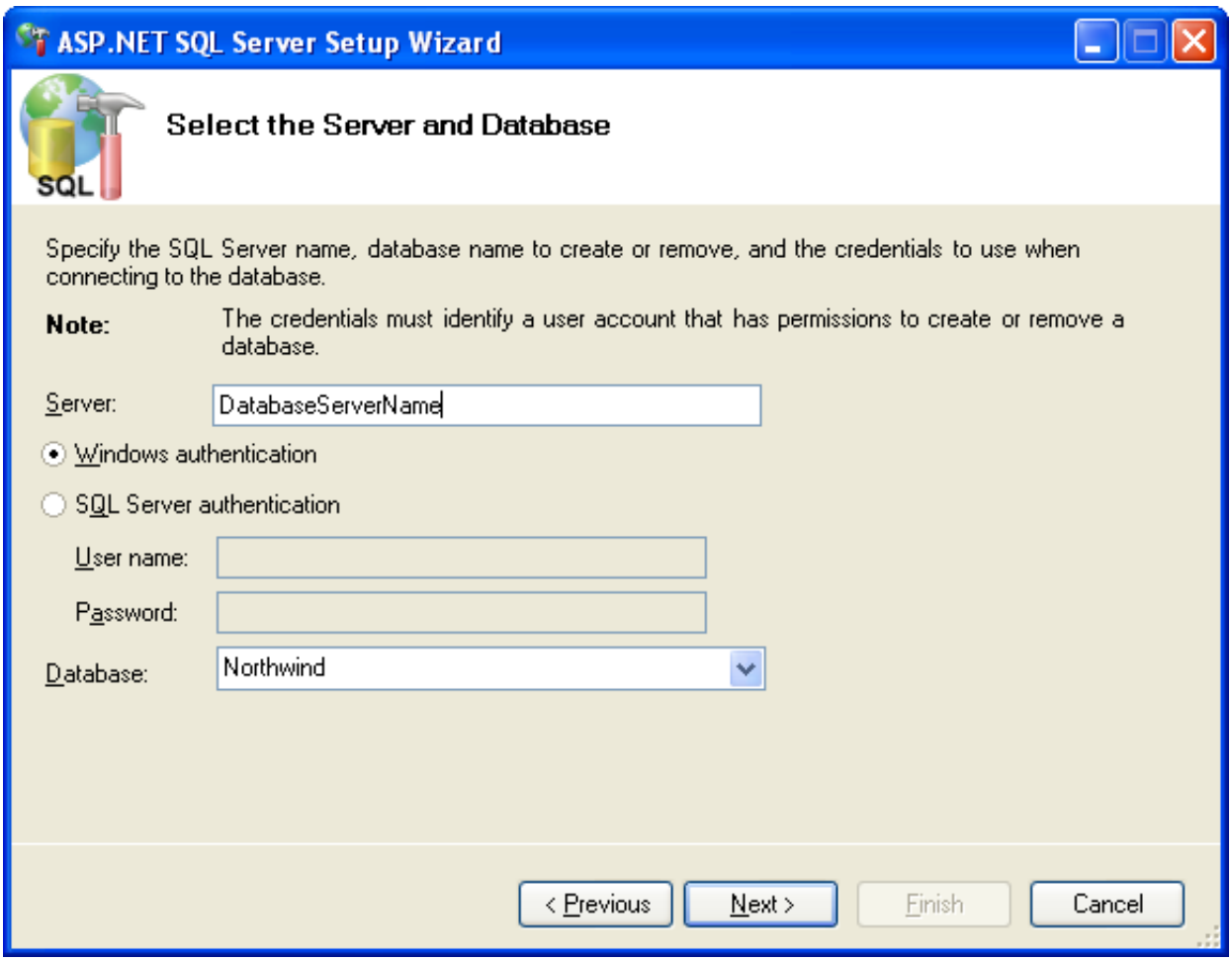
Applying the Membership-Related Schema to an Existing Database

Applying the membership-related schema to an existing database is an easy task with the ASP.NET SQL

Server Registration Tool (`aspnet_regsql.exe`). Simply drop to the command-line, navigate to `%WINDOWS%\Microsoft.NET\Framework\v2.0.50727` and run `aspnet_regsql.exe`. This will bring up the ASP.NET SQL Server Setup Wizard. From the wizard you can either configure a SQL Server database for application services (i.e., add the membership-related schema) or you can remove this schema from a database that no longer needs it.



On the next screen you are prompted to select the database for which the membership-related schema should be added (or removed).



After you have specified that database, click Next to review the changes that will be made. After previewing the changes click Next to install the schema. This will add the tables, views, and stored procedures needed by the SQL Server-based membership providers.

What this ASP.NET SQL Server Registration Tool does is run the SQL script found at `%WINDOWS%\Microsoft.NET\Framework\v2.0.50727\InstallCommon.sql` with some slight modifications. The `%WINDOWS%\Microsoft.NET\Framework\v2.0.50727\InstallCommon.sql` script, as-is, tries to work with a database named `aspnetdb`. The ASP.NET SQL Server Registration Tool essentially runs this script but instead uses the database name specified in the wizard. The `InstallCommon.sql` installs *all* of the membership-related services: Membership, Roles, Profile, Personalization, and so on. If you want to install the schema for just a subset of these features there are more specific scripts in `%WINDOWS%\Microsoft.NET\Framework\v2.0.50727\`.

This script attempts to connect to the specified database name (either `aspnetdb` or the name provided in the wizard). If it can't find the database in the server's `sysdatabases` table, it attempts to create a new database with the specified name. It then adds the assorted tables, views, and stored procedures.

Running the ASP.NET SQL Server Registration Tool from the Command-Line

In addition to providing a wizard, the ASP.NET SQL Server Registration Tool also can be invoked from the command-line. Navigate to `%WINDOWS%\Microsoft.NET\Framework\v2.0.50727` and run:

```
-- To use Windows Authentication (i.e., a "trusted connection"), use:
aspnet_regsql.exe -S <server> -E -d <database> -A all
```

```
-- To use SQL Server credentials (a UserID and Password), use:
aspnet_regsql.exe -S <server> -U <login id> -P <password> -d <database> -A
all
```

The `-A all` installs the schema for *all* of the membership-related services. You can specify the precise set of membership-related features to install using `-A list`, like `-A mr` to install the schema for just membership and roles. Run `aspnet_regsql.exe -?` to see a full list of the command-line options.

Installing the Membership-Related Schema on a SQL Server 2005 Express Edition Database in `App_Data` If your application's existing database is a SQL Server 2005 Express Edition database in the `App_Data` folder you may be wondering how, exactly, to access it through the ASP.NET SQL Server Registration Tool. What do you use for the server name, credentials, and database name? I'll be the first to admit that I'm no expert when it comes to database administration, but here's what I've picked up while tinkering with the Express Edition:

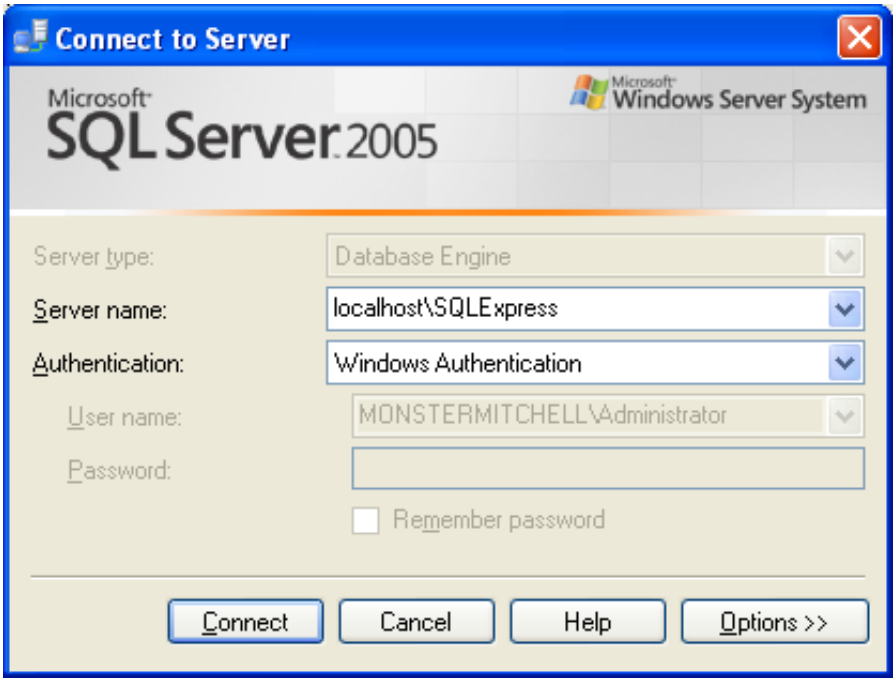
- The Server name is `localhost\InstanceName`; by default, SQL Server 2005 Express Edition's *InstanceName* is `SQLExpress`, so the Server name to use is likely `localhost\SQLExpress`.

- By default SQL Server 2005 Express Edition is setup to accept Windows Authentication only with access granted to any user on the system. So unless you've changed this, just leave the authentication option set to Windows Authentication.

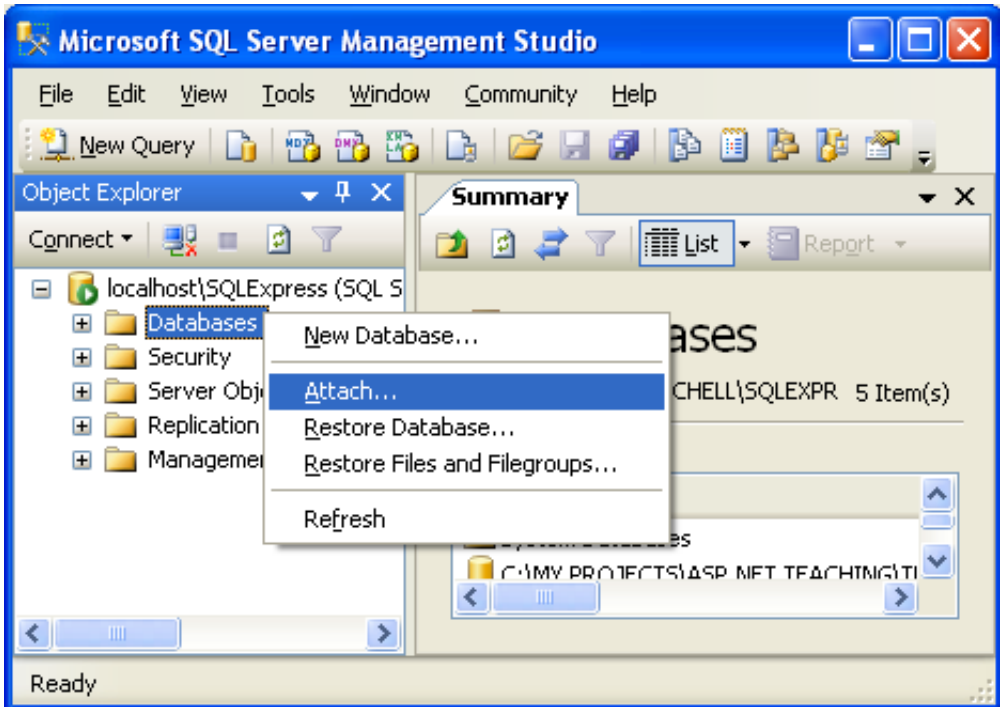
- The database name is the tricky part. From what I can ascertain, when creating a SQL Server 2005 Express Edition database in the `App_Data` folder the database's name is the fully qualified path to the database. That is, if your database is in `C:\Home\Websites\PhotoAlbum\App_Data\Photos.mdf` then the database's name is `C:\Home\Websites\PhotoAlbum\App_Data\Photos.mdf`. However, the database is likely *not* attached to the SQL Server 2005 Express Edition instance. We need to attach it and give it a name in the database server (such as `Photos`) which we can then reference in the ASP.NET SQL Server Registration Tool.

Giving a name to an existing SQL Server 2005 Express Edition database is most easily done by downloading the free SQL Server 2005 Management Studio Express Edition, from which you can work with SQL Server 2005 Express Edition through a GUI application.

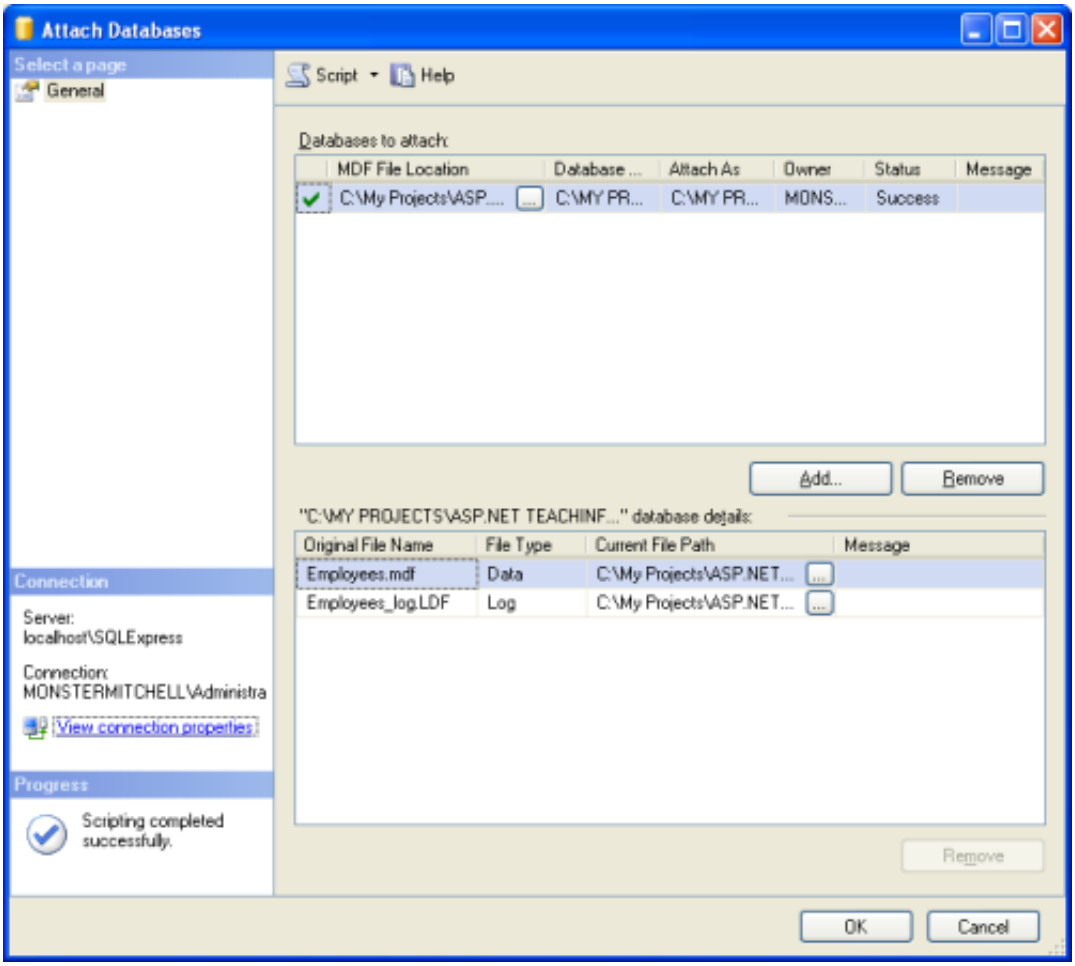
If you have SQL Server 2005 Standard Edition or higher installed on your computer then you may already have the Management Studio for SQL Server 2005 installed, in which case you *cannot* install the Express Edition version of Management Studio. However, you can connect to the SQL Server 2005 Express Edition instance by specifying the server name correctly in the connection screen: `localhost\SQLExpress` (or whatever it may be for your setup).



From Management Studio after connecting to a database server you can attach a database by right-clicking on the Databases folder and choosing Attach.



You can then select the file to attach from your hard drive. The database name will be the *original* full path to the database file when it was created. I say *original* because if you have moved the database from one directory to another, the database name remains the name of the path when the database file was created. You can change the name of the attached database by scripting the action (click the Script button) and modifying the script to use a different database name. Alternatively, you can rename a database in Management Studio by right-clicking on the database and choosing Rename.



Once you have the database attached you can specify it by that name in the ASP.NET SQL Server Registration Tool, either through the drop-down list in the wizard or via the command-line. Speaking of the command-line, you can also attach a database to the SQL Server 2005 Express Edition instance via the `sqlcmd` command-line program, if Management Studio and pointing and clicking just aren't your thing. See my blog entry *Working With SQL Server 2005 Express Databases* for more information.

Telling the Membership Providers to Use Your Application's Existing Database

With the membership-related schema merged into your application's existing database, you're ready to start having the membership-related queries directed to your database as opposed to `ASPNETDB.MDF`. However, the default SQL Server Membership providers - `SqlMembershipProvider`, `SqlRoleProvider`, `SqlProfileProvider`, and so on - all want to use `ASPNETDB.MDF` and will send all queries there. To remedy this, we need to add our own provider instances in `Web.config` that use our existing database and make these providers the default Membership providers.

As we saw in Part 1 of this article series, the provider information can be configured through `Web.config`. To customize both the membership and roles providers to use our existing database for membership queries rather than `ASPNETDB.MDF`, we'd use something like:

```
<configuration>
  <connectionStrings>
    <add name="MyDB" connectionString="..." />
  </connectionStrings>
  <system.web>
    ... authentication & authorization settings ...

    <roleManager enabled="true"
                 defaultProvider="CustomizedRoleProvider">
```

```
      <providers>
         <add name="CustomizedRoleProvider"
              type="System.Web.Security.SqlRoleProvider"
              connectionStringName="MyDB" />
      </providers>
    </roleManager>

    <membership defaultProvider="CustomizedMembershipProvider">
      <providers>
         <add name="CustomizedMembershipProvider"
              type="System.Web.Security.SqlMembershipProvider"
              connectionStringName="MyDB" />
      </providers>
    </membership>

  </system.web>
</configuration>
```

These settings create custom providers for the membership and roles systems that use the existing SQL-based providers but customize them to use the connection string named `MyDB`, which would point to the application's existing database. With this change all membership-related queries would be routed to the existing database.

Conclusion

When adding membership support to an application, often we need to take the membership schema and apply it to an existing database. This can be accomplished through the ASP.NET SQL Server Registration Tool, either through a graphical wizard or directly from the command-line. Once the membership schema has been merged with the application's existing data model, the final step is to customize the providers so that they use the existing database as opposed to the default one (`ASPNETDB.MDF`).

Happy Programming!

- By Scott Mitchell

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 3
Article Author:	Scott Mitchell
Published Date:	April 5, 2006
Article URL:	http://www.4GuysFromRolla.com/articles/040506-1.aspx